Hierarchical and Distributed Machine Learning Inference Beyond the Edge

Anthony Thomas*, Yunhui Guo*, Yeseong Kim*, Baris Aksanli†, Arun Kumar*, Tajana S. Rosing*
*UC San Diego
La Jolla, CA 92093
{ahthomas,yug185,yek048,arunkk,tajana}@eng.ucsd.edu
†San Diego State University
San Diego, CA 92182
baksanli@sdsu.edu

***Abstract*—Networked applications with heterogeneous sensors are a growing source of data. Such applications use machine learning (ML) to make real-time predictions. Currently, features from all sensors are collected in a centralized cloud-based tier to form the whole feature vector for ML prediction. This approach has high communication cost, which wastes energy and often bottlenecks the network. In this work, we study an alternative approach that mitigates such issues by "pushing" ML inference computations out of the cloud and onto a hierarchy of IoT devices. Our approach presents a new technical challenge of "rewriting" an ML inference computation to factor it over a network of devices without significantly reducing prediction accuracy. We introduce novel exact factoring algorithms for some popular models that preserve accuracy. We also create novel approximate variants of other models that offer high accuracy. Measurements on a common IoT device show that energy use and latency can be reduced by up to 63% and 67% respectively without reducing accuracy relative to sending all data to the cloud.**

***Keywords*—machine learning; IoT; edge computing; energy efficient computing**

I. Introduction

The rapid growth in the number and variety of networked sensors, collectively called the Internet of Things (IoT), is causing a proliferation of data in a wide range of sensing applications. Many providers of IoT systems are turning to sophisticated machine learning (ML) techniques to analyze this data. For example, Samsung now offers a robust line of sensors designed for smart-home applications which integrate with (predominantly) cloud based processing applications [1]. Similar product lines are offered by other major companies as well [2], [3]. These deployments are still largely reliant on the cloud for data processing with local processing on devices typically supported only for relatively simple tasks which do not require complex algorithms [4].

In this work, we consider how the inference computation for complex, general purpose, ML algorithms can be efficiently implemented in *heterogeneous* IoT applications which require input from sensors gathering fundamentally different features. As a running example, in Figure 1B, we consider an application for predicting aggregate power demand from real-time data on power use by individual appliances in a pair of homes. Such applications have recently been proposed as a more accurate alternative to aggregate data gathered from power use meters for energy demand forecasting [5]. In the currently dominant "monolithic" approach, illustrated in Figure 1B(i), each feature is communicated to the cloud where it is fed into an ML model to predict energy demand.

In this context, ML algorithms are typically trained periodically offline and then deployed for predictions on live streaming data. While online, supervised, training of models in sensor networks has garnered significant research interest in recent years, it remains a challenging problem due to the need for labeled data and high communication cost for many practical learning algorithms. Furthermore, inference occurs with far greater frequency than learning and represents a significant computational burden in its own right [6]. Accordingly, our primary focus here is on inference.

The monolithic approach to inference has at least three key issues. First, excessive communication reduces battery life. Second, in some applications, the volume of data transmitted can saturate the network, either crippling the system or leading to increased latency [7]. Third, some applications have privacy constraints that might be difficult to satisfy, e.g., a smart-home user may not want their raw data to leave the home.

To mitigate these issues, recent work has proposed a modular "hierarchical" approach for heterogeneous IoT applications that pushes computation out of the cloud and onto IoT devices [7]. This approach introduced software modules, referred to as "context engines," which run on IoT devices and compute an aggregated representation of input data. Input to these context engines may come from sensors connected by a short-range link (e.g., Bluetooth) or the output of other context engines. Thus, context engines can be organized in a multi-layer hierarchy that successively abstracts the raw data, as illustrated by Figure 1A. Devices running context engines have CPUs and bi-directional wireless communication - a representative example is the Raspberry Pi3 [8].

While work in [7] has shown the hierarchical model to be effective for reducing communication, it complicates ML application deployment. In the hierarchical setting, the cloud no longer has access to the entire feature vector - only the aggregated output of lower tiers in the hierarchy. This means that each context engine can only compute a *partial inference result* based on the data it can access locally. The cloud must then compose these partial results to produce the final prediction. We refer to this process as "hierarchy-aware inference" and provide an example in Figure 1B(ii). Under these constraints, the inference computation of many popular ML algorithms either cannot be expressed at all or requires

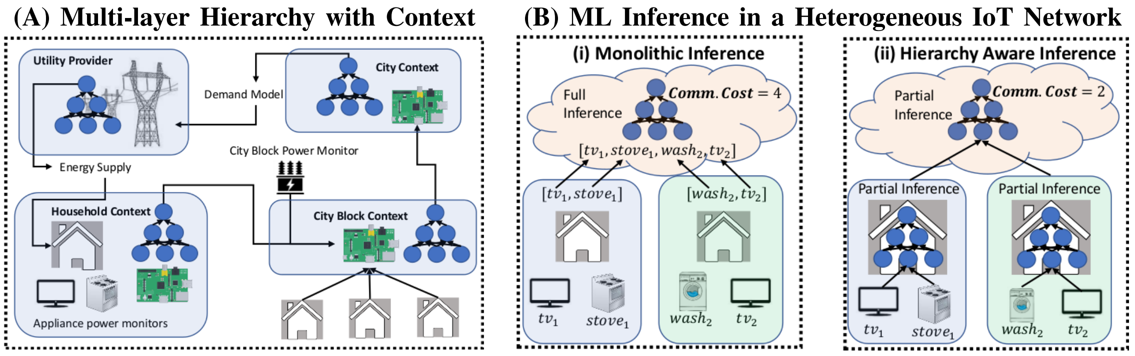

Fig. 1. (A) An example heterogeneous IoT application for predicting the power consumption of a neighborhood. (B.i) Simplified example of monolithic inference in a 2-tier hierarchy with appliance-specific power use features. In this example, the number of features communicated goes down from 4 to 2.

potentially burdensome levels of communication.

In this work we introduce several novel techniques to perform inference for complex, general purpose, ML algorithms in hierarchical IoT applications. We consider generalized linear models, decision trees, boosted trees and random forests, along with multilayer-perceptron (MLP) and recurrent (LSTM) neural networks. We introduce a novel hierarchy-aware inference algorithm for decision trees and tree ensembles which minimizes communication cost with no loss in accuracy. We additionally present several "hierarchy-aware" neural network architectures which enable users to trade off between communication cost and accuracy. Our goal is to reduce energy use and latency by decreasing the volume of data which must be communicated to perform an inference computation. Using measurements gathered from a real-world deployment on a popular IoT device, we show that our approach leads to substantial - up to 67% - reductions in energy use and latency without sacrificing accuracy.

II. Related Work

General trade-offs between computing in the cloud or beyond the edge are well studied in the IoT community [9], [10], [11], [12]. However, prior work has primarily focused on choosing between a lightweight model which can be fully locally computed on an IoT device and a more computationally demanding (but more accurate) model running in the cloud. In the context of our example from Figure 1B, this would correspond to treating both homes as independent data sources and choosing between a simple model which could be computed directly in each home and a more accurate model running in the cloud. However, we assume a full inference result is not possible locally and consider inference computations which must be distributed across devices. Dependence between sensors was addressed by [7], [13] but only for simple linear aggregation functions. In contrast, our work is more general (covering nonlinear models), and we propose novel hierarchy-aware implementations for more complex ML models.

Recent work in [14], [15] studied how to pick the layer at which a deep neural network's computation should be split between the cloud and beyond the edge to reduce energy use and latency, but they focused on image processing applications and did not address hierarchical architecture or statistical dependence between data sources. The most closely related work to ours is [16], which proposed a "hierarchy-aware" convolutional neural network for image classification. Their work assumes *homogeneous* sensors (video cameras), which implies that each device can still perform full inference, unlike our setting. Additionally, they pass data up the hierarchy only to improve accuracy, while fully device-local inference is impossible in our setting.

Similarly, [17], [18] introduced a technique known as "federated learning" in which a centralized model is trained over a distributed collection of devices (e.g., smartphones) using a model averaging procedure. Returning to our running example, this work assumes that a model can be "cloned" to each house and trained on locally available data. Each house periodically communicates its local model to the cloud which computes a weighted average over all models and then transmits back updated weights. However, this work is not applicable to our setting as it assumes the availability of labeled data at each context engine and that each device senses the same feature set. Furthermore, the goal of federated learning (as we understand it) is ultimately to generate local "on-device" predictions while we focus on groups of sensors collaborating to generate predictions in the cloud. Overall, our work differs from the above in that we study ML deployments which require input from a distributed collection of *heterogeneous* sensors in a *multi-layer* hierarchy in which nonlinear dependencies are assumed to exist between data gathered by different nodes.

III. Hierarchy-Aware ML Inference

We now present our novel hierarchy-aware implementations of several popular models. We consider an IoT deployment consisting of a set of m devices which sense a total of $d \gg m$ features. ML inference is a function $Y = f(\boldsymbol{x})$, where $\boldsymbol{x}$ is a *feature vector* of length d. The elements of $\boldsymbol{x}$ may be real numbers or integers. We denote by $E_1, \ldots, E_m$ the devices at or beyond the edge. In general, $\boldsymbol{x}$ is only available in the cloud and a device E_i only has access to a (disjoint) subset $\boldsymbol{x}_i \subset \boldsymbol{x}$ of the features.

Overall, our goal is to reduce the volume of data (e.g. bytes) which must be communicated to compute $f(\boldsymbol{x})$ without significantly reducing accuracy or introducing high additional computational cost on the IoT devices. Our approach is based on finding a *decomposition* of $f(\boldsymbol{x})$ into a set of functions which can be computed using only the data available locally to each device E_i. For example, in a two layer hierarchy this amounts to finding a decomposition $f(\boldsymbol{x}) \approx g_0(g_1(\boldsymbol{x}_1), \ldots, g_m(\boldsymbol{x}_m))$. Communication is minimized when the output of each g_i is a single number and is lossless if the decomposition holds with equality. We remark that our approach is *complementary* to techniques which reduce communication by quantization or precision reduction. We begin our discussion by considering generalized linear models (GLMs) as a simple illustrative example and then move on to tree based models and neural networks which are of greater practical interest.

A. Generalized Linear Models

GLMs model the data using a hyperplane $\boldsymbol{w} \in \mathbb{R}^d$ that separates the classes (for classification) or predicts a continuous target (for regression). The inference computation for a GLM simply computes an inner product $u = \boldsymbol{w}^T\boldsymbol{x}$ and then applies a scalar valued function $\sigma(u)$ to the result.

Hierarchy-Aware GLMs. On device E_i that senses features $\boldsymbol{x}_i \subset \boldsymbol{x}$, we pre-store the weights corresponding to the locally available features and compute $u_i = \boldsymbol{w}_i^T\boldsymbol{x}_i$ for each inference request. The intermediate variable u_i is then sent to the cloud which finishes the inference computation by computing $\sigma(\sum_{i=1}^m u_i)$. Since sums are associative, there is no loss in accuracy relative to the full communication setting. Furthermore, since each device simply transmits a single scalar value, the communication cost attains the lower bound of m.

While GLMs are simple to deploy in the hierarchical setting, they are of limited usefulness as learning algorithms because they can only model *linear* relationships in the data. Furthermore, standard techniques to address this limitation, such as polynomial regression or support vector machines, incur prohibitively high communication cost in the hierarchical setting. We now present novel hierarchy-aware inference implementations for two popular and powerful ML models: decision trees and neural networks.

B. Decision Trees and Random Forests

A decision tree recursively partitions the d-dimensional feature space into hyperrectangles and approximates the data generating process using a constant for each hyperrectangle [19]. This partitioning is captured using a tree data structure. An internal node has a Boolean expression of the form $z < \alpha$, where z is an individual feature in $\boldsymbol{x}$ having domain $\mathcal{D}_z$. The value $\alpha \in \mathcal{D}_z$ is a "split-point" learned from training data. A leaf node carries the value predicted by the tree. Inference in a decision tree traces a path from root to leaf based on the sequential evaluation of the Boolean expression at each node.

Hierarchy-Aware Decision Trees. At first glance, it might seem impossible to perform decision tree inference in our setting: an arbitrary inference path from root to leaf will involve multiple features in $\boldsymbol{x}$, not all of which are available on a single device - as in Figure 2(A). However, we can devise a efficient inference procedure by observing that prediction does not require exact knowledge of feature values - only the partition of feature space an example falls under. Our procedure has two phases: an offline pre-processing phase and an online inference phase. An example is given in Figure 2(B).

In the offline pre-processing phase we are given a learned decision tree T. For each feature $z \in \boldsymbol{x}$ we extract the set of all *split points* on $z \in T$ and store them in a *sorted* array $\boldsymbol{v}(z) = [\alpha_1, \alpha_2, ..., \alpha_p]$. This array defines intervals that *partition* $\mathcal{D}_z$ into p disjoint intervals - i.e. $\boldsymbol{v}(z)_i \Rightarrow \alpha_{i-1} \leq z < \alpha_i$. Boosting and random forests (see: [20], [21]) can be handled by taking $\boldsymbol{v}(z)$ to be the common refinement of the $\boldsymbol{v}(z_j)$ corresponding to each individual tree T_j. We assign *sequential integers* (0 to $p-1$) to these intervals, and set an arbitrary (say lexicographical) ordering for the arrays corresponding to each feature. Each device, along with the cloud, stores a copy of the interval arrays corresponding its locally available features.

The online inference phase proceeds as follows. For a device E_i and for each feature $z_j \in \boldsymbol{x}_i$, determine the interval partition number/index z_j falls into using binary search over $\boldsymbol{v}(z_j)$. Denote the resulting indices by $\beta_1, \ldots, \beta_k$. We can then represent this tuple of indices as a *single scalar value* using the equation $g = \sum_{i=1}^k \beta_i \prod_{j=i+1}^k p_j$. An example is presented in Figure 2(B). This value encodes the region of the feature space the example falls under and is transmitted to the cloud.

The cloud simply inverts the computations performed by each device E_i using standard modular arithmetic to recover the indices corresponding to the interval into which each feature $z \in \boldsymbol{x}$ falls and samples a value $\tilde{z}$ from this interval. The value $\tilde{z}$ is then used in a conventional inference computation as in Figure 2(A). Thus, the cloud does not necessarily recover the true value of z, but it recovers a value lying the *same partition* of $\mathcal{D}_z$ and hence returns the *same prediction*. Since each device only needs to communicate a single value, the communication cost attains the lower bound of m. A caveat to this statement is that the bit-length required to represent the value communicated by E_i may exceed a standard floating point number (64 bits) if the number of split points in the x_i is large (i.e. the tree is deep). In our experimental evaluation in section IV, we find that 64 bits is typically sufficient.

C. Neural Networks

We now to turn to the multilayer-perceptron (MLP), which is a powerful generalization of the GLM capable of automatically "learning" nonlinear structure in data. As shown in Figure 3(A), the MLP can be seen as a directed graph - the internal nodes are called "hidden units." The first hidden layer (consisting of h hidden units) computes $g(\boldsymbol{x}) = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$, where $\boldsymbol{W}$ is an $h \times d$ matrix of weights and $\boldsymbol{b}$ in an $h \times 1$ vector of constants both of which are fixed for inference. The cost of MLP inference in the hierarchical setting is mh. Thus, for some values of h this may be communication saving, but generally will not be as h is typically in the dozens or even hundreds. Along with the MLP we also consider "Projection Pursuit Regression" (PPR) which is similar to a single layer

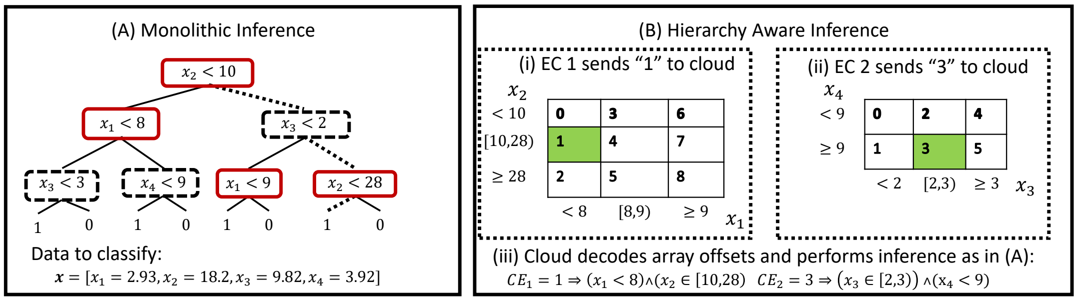
(A) Monolithic Inference
$x_2 < 10$
$x_1 < 8$
$x_3 < 2$
$x_3 < 3$
$x_4 < 9$
$x_1 < 9$
$x_2 < 28$
1 0 1 0 1 0 1 0
Data to classify:
$x = [x_1 = 2.93, x_2 = 18.2, x_3 = 9.82, x_4 = 3.92]$
(B) Hierarchy Aware Inference
(i) EC 1 sends "1" to cloud
(ii) EC 2 sends "3" to cloud
x_2
< 10
[10,28)
≥ 28
< 8
[8,9)
≥ 9
x_1
x_4
< 9
≥ 9
< 2
[2,3)
≥ 3
x_3
(iii) Cloud decodes array offsets and performs inference as in (A):
$CE_1 = 1 \Rightarrow (x_1 < 8) \wedge (x_2 \in [10,28)$ $CE_2 = 3 \Rightarrow (x_3 \in [2,3)) \wedge (x_4 < 9)$

Fig. 2. (A) Illustration of standard monolithic inference for a simplified binary classification decision tree. (B) Illustration of our hierarchy-aware inference

MLP but replaces the fixed activation function with a spline and typically requires a much smaller h [22]. The vanilla MLP can be extended in a straightforward manner to support analysis of time-series data through the popular "long-short-term memory" (LSTM) network architecture [23]. For our purposes, it suffices to know that $f(\boldsymbol{x})$ for an LSTM computes 4 matrix-vector products in the first layer, each of which has the same basic computation as the single g in an MLP.

Hierarchy-Aware Neural Networks. Our modified MLP architecture is illustrated by Figure 3(C). Our approach is to "wire in," *a priori* of training, a partial connectivity structure based on the IoT application's network imposed partitioning of $\boldsymbol{x}$. On each device E_i^1 in the first layer of the hierarchy, we maintain a *local MLP* $\boldsymbol{u}_i^1 = g_i^1(\boldsymbol{x}_i)$ which only considers the locally available features $\boldsymbol{x}_i$. As shown in Figure 3(C), the output of each local network in the first layer of the hierarchy is used as input by one or more local MLPs running on devices in the second layer. This process is repeated recursively and culminates in a fully connected MLP running in the cloud. More formally, the i-th device in the j-th layer of the hierarchy runs an MLP $u_i^j = g_i^j(\boldsymbol{u}_i^{j-1})$, where $\boldsymbol{u}_i^{j-1}$ denotes locally available input to the i-th device - either raw features or the output of previous layers. Because the model is simply a composition of local MLPs the entire model can be jointly trained end-to-end using standard back-propagation.

The complexity of the relationship between devices which can be modeled is controlled by the output dimension κ of each local MLP - a larger value of κ will allow for more complex relationships, but will incur additional communication cost and increase the risk of overfitting. Because the MLP is a non-convex optimization problem, there is no analytic means for selecting κ which must be chosen using standard hyperparmeter tuning techniques (e.g. grid-search)[19]. This approach generalizes naturally to recurrent neural network (RNNs) architectures such as the popular LSTM architecture - shown in Figure 3(D) and (E).

IV. Experimental Evaluation

We now evaluate our proposed hierarchy-aware ML inference techniques on a suite of three real-world datasets from heterogeneous IoT use cases covering both classification and regression. Our goal is to verify that our hierarchy-aware approaches do not reduce accuracy relative to models which allow for unlimited communication. We selected benchmark tasks which were both reflective of common IoT use-cases and for which we expect modeling nonlinear relationships to yield increased accuracy. The datasets are summarized in Table I.

Task/Dataset	Type	n	d	m
Urban Energy Demand	Reg.	$125,549$	387	52
Human Activity	Class. (5)	$804,228$	52	4
Server Performance	Reg.	$24,729$	60	5

TABLE I
DATASET STATISTICS. n IS THE NUMBER OF EXAMPLES, d IS THE NUMBER OF FEATURES, AND m IS THE NUMBER OF DEVICES (CONTEXT ENGINES).

The first dataset is a regression task which seeks to predict aggregate neighborhood level power consumption from individual appliances which were instrumented with power use sensors. We use a commercial dataset provided by "Pecan Street Measurements" for this task [24]. The second dataset is a classification task which infers the label for one of five everyday activities (e.g. walking, running, cleaning, etc...) using data gathered from three IMU sensors mounted at the chest, ankle and wrist, along with a heart rate monitor. We use the PAMAP2 dataset for this task which is publicly available from the UCI machine learning repository [25]. The third and final task is a regression task in which we attempt to predict the performance of a program on a computer in a cluster using data gathered from "performance measurement units" (PMU) (e.g. cache hits, number of executed instructions, etc...). We generated this data ourselves by instrumenting six servers to record twelve PMU events as described in [26] while running a set of twenty-six Apache Spark applications.

A. *Results and Discussion*

Figure 4 presents the overall accuracy and communication cost for each model we compare. Accuracy and communication cost are reported relative to the linear model which represents the most naive hierarchy-aware implementation. For example, in Figure 4(A), the square point at $(2, 0.85)$ indicates that accuracy is improved by 15% relative to the linear model but the total bytes communicated are doubled. Our goal is to exceed the accuracy of the linear model by handling nonlinear relationships between features associated with different context engines without substantially increasing communication cost. For models which allow us to vary the number of outputs

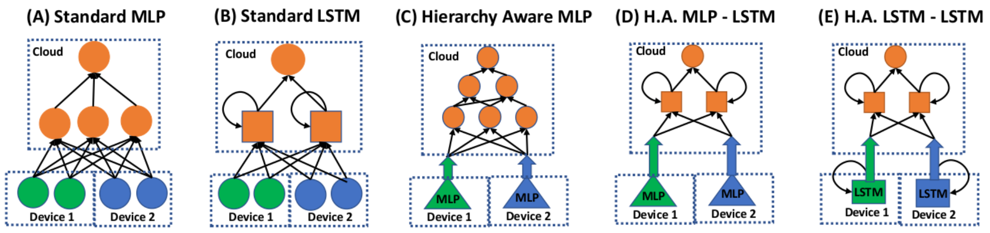

Fig. 3. Simplified illustrations of various neural network architectures considered in this paper. "H.A." stands for "Hierarchy-Aware." Orange nodes reside in the cloud; green nodes, on device 1; blue nodes, on device 2. (A) Standard fully connected MLP - each device sends 1 number per orange node in the first hidden layer. (B) Standard fully connected LSTM - each device sends four numbers per orange node; (C,D,E) Our HA MLP implementations. In C, D and E each device always sends one number.

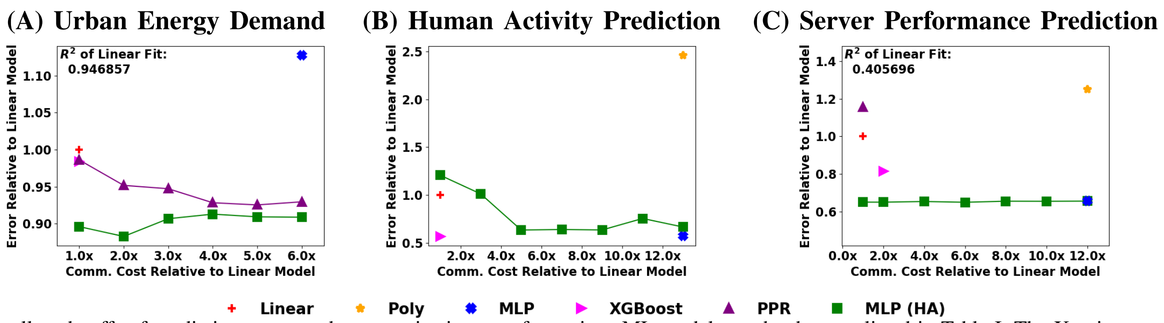

Fig. 4. Overall trade-offs of prediction errors and communication cost for various ML models on the datasets listed in Table I. The X-axis reports prediction errors relative to the linear model. The Y-axis reports the total bytes of data communicated relative to the linear model. Linear is a GLM without feature interactions; Poly is a GLM with order 2 interactions (not feasible for panel A). MLP is the monolithic architecture. MLP (HA) is our hierarchy-aware MLP.

from each context engine, we try values between 1 and $|x_i|$ - these are the interior points in the plots.

While no single model dominates all three tasks, we found the hierarchy-aware MLP to generally deliver strong performance. On all three tasks, we found that the LSTM based models (a type of recurrent neural network) overfit in spite of substantial hyperparameter tuning and so were omitted from results to avoid obfuscating trends in the other models. LSTMs are capable of modeling complex temporal structure in data and are likely simply overkill for these relatively simple time-series problems. Similarly, while PPR performed well on the power prediction task, we found it overfit significantly on the server performance task.

Overall, we see that for each dataset, at least one of our hierarchy-aware ML models attains comparable or better accuracy relative to the best monolithic ML model, while yielding far lower communication cost. On all three datasets, we find that modeling nonlinear relationships between the data gathered by different devices in the hierarchy substantially improves prediction accuracy. Thus, these results validate our core claim in this paper: *our hierarchy-aware techniques for ML inference can reduce communication costs substantially, while still yielding high accuracy.*

B. Implementation and Measurement on RPI3

To understand how our proposed methods would benefit real world deployments, we implemented our proposed hierarchy-aware inference methods for XGBoost and the MLP (with a compression factor of $\kappa = 1$) on a Raspberry Pi3 (RPi3) [27]. The RPi3 communicates with another Pi3 acting as a server using the MQTT protocol which is widely used for IoT applications [28].

To simulate a variety of network speeds, we use the Linux Kernel's traffic controler (`tc`) to regulate the server's bandwidth. TC restricts incoming traffic by dropping a fraction of packets which is reflective of congested networks. To implement XGBoost, we learned a tree offline and extracted the underlying data structure to identify the split points used by each feature in all trees. We then implemented the algorithm as described in Section III-B. We measure latency as the total time required to perform computation (relevant in the hierarchy-aware case only) and transmit data over MQTT. Energy use is measured as the current drawn by the device during computation and communication. We tested our approach on the Human Activity dataset [25], varying the number of inference requests from 200 to 400 per second. We report results for communication bandwidths of 200, 700 and 5000 kbps, which correspond to typical bandwidths of constrained Bluetooth, Bluetooth, and WiFi, respectively [29].

Figure 5 shows that the proposed methods can significantly reduce energy use and latency, especially when network bandwidth is low. The reductions in latency and energy consumption are up to 67% and 63% respectively. The sharp increases in both plots at 220, 260 and 400 samples per second are driven by increased communication time in the monolithic case. We attribute these increases to overhead in the TCP layer resulting from packet loss. We note that the hierarchy-aware MLP and XGBoost exhibit similar trends for both energy and latency reduction, indicating the extra computation cost incurred by

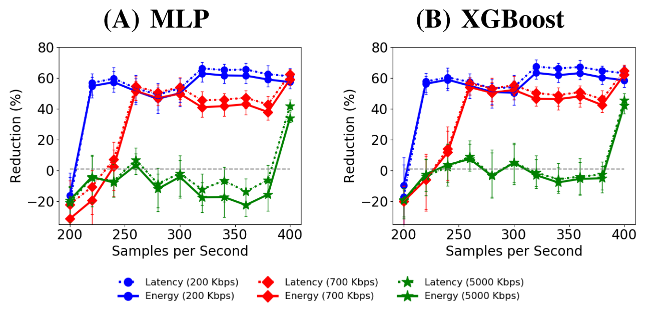

Fig. 5. Percentage reduction of latency and energy consumption from monolithic to our hierarchy-aware ML inference on the Human Activity dataset. "Samples per second" is the number of inference requests received per second. Solid lines indicate energy use while dashed indicate latency.

the device-local MLP is not significant. As expected, both model types see largest efficiency gains when the network bandwidth is more limited.

V. Conclusion

As the use of ML in heterogeneous Internet of Things applications grows, efficient management of IoT data for ML inference has become a pressing challenge. In this work we present approaches which factor inference computations for several powerful and widely used ML models over a hierarchy of IoT devices. By computing partial inference results locally and transmitting only aggregated data up the hierarchy to the cloud we reduce the overall cost of data movement. Using experiments on three real world datasets and measurements on a commonly used IoT device, we show our methods can substantially reduce energy use and latency without losing accuracy.

VI. Acknowledgements

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also by NSF grants 1730158 and 1527034.

References

[1] Samsung Corporation, "SmartThings Classic Developer Documentation," https://docs.smartthings.com/en/latest/index.html#smartthings-classic-developer-documentation, 2019.

[2] Amazon Corporation, "Amazon Alexa SmartHome," https://www.amazon.com/b?node=17934679011, 2019.

[3] Google Inc., "Google Home," https://store.google.com/us/product/google_home_smart_home?hl=en-US, 2019.

[4] Samsung Corporation, "Samsung Smartthings: Local Processing," https://support.smartthings.com/hc/en-us/articles/209979766-Local-processing, 2019.

[5] B. Aksanli, "Accurate and data-limited prediction for smart home energy management," *Proceedings of the ASME 2018 Power and Energy Conference*, 2018.

[6] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, 2017, pp. 613–627.

[7] J. Venkatesh, C. Chan, A. S. Akyurek, and T. S. Rosing, "A modular approach to context-aware iot applications," in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, April 2016, pp. 235–240.

[8] Raspberry Pi Foundation, "Raspberry Pi 3 specifications," https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[9] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, Nov 2015, pp. 73–78.

[10] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[11] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, 2015.

[12] P. Zhang, M. Zhou, and G. Fortino, "Security and trust issues in fog computing: A survey," *Future Generation Computer Systems*, vol. 88, pp. 16–27, 2018.

[13] H. Grunert and A. Heuer, "Rewriting complex queries from cloud to fog under capability constraints to protect the users' privacy," *Open Journal of Internet Of Things (OJIOT)*, vol. 3, no. 1, pp. 31–45, 2017.

[14] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. ACM, 2017, pp. 615–629.

[15] M. F. A. Jong Hwan Ko, Taesik Na and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," https://arxiv.org/pdf/1802.03835.pdf, 2018, arXiv Preprint.

[16] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 328–339.

[17] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[18] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, 2017.

[19] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1.

[20] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[21] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. ACM, 2016, pp. 785–794.

[22] J. H. Friedman and W. Stuetzle, "Projection pursuit regression," *Journal of the American statistical Association*, vol. 76, no. 376, pp. 817–823, 1981.

[23] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[24] "Pecan street dataport," https://dataport.cloud/, 2018.

[25] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *2012 16th International Symposium on Wearable Computers*, June 2012, pp. 108–109.

[26] Y. Kim, P. Mercati, A. More, E. Shriver, and T. Rosing, "P4: Phase-based power/performance prediction of heterogeneous systems via neural networks," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2017, pp. 683–690.

[27] W. Cui, Y. Kim, and T. S. Rosing, "Cross-platform machine learning characterization for task allocation in iot ecosystems," in *Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual*. IEEE, 2017, pp. 1–7.

[28] MQTT, "Mqtt protocol." [Online]. Available: http://mqtt.org/

[29] R. Friedman, A. Kogan, and Y. Krivolapov, "On power and throughput tradeoffs of wifi and bluetooth in smartphones," *IEEE Transactions on Mobile Computing*, vol. 12, no. 7, pp. 1363–1376, 2013.